



Sybase性能优化知识库



写在前面

- Sybase针对于性能的配置包含很多层次，有很多细节。
- 此文档并不是要对所有的性能问题进行囊括，只是作为Sybase的开发小组人员进行内部培训而用
- 比较宝贵的是，文档中的一些内容是从其他官方资料中找不到的，而是来源于Sybase技术工程师的建议
- 文档比较简单的语言进行描述，让大家能听懂，并且不需要涉及深入的技术细节。但是**鼓励大家去学习专业的技术文档，提高专业技能**
- 希望大家能够不断丰富此文档，作为knowhow进行保存，也作为今后的新人培训资料



目录

- 统计信息
- 存储优化
- 临时数据库
- 索引
- 性能调查工具
- 索引与画面设计



关于统计信息

- 概念
- 方法
- 策略



统计信息的基本概念

- 在访问表时，Sybase的**优化程序** 利用**统计信息**来决定如何访问以达到开销最低。如果说表的索引是公路，统计信息就是各条公路的里程记录，优化程序会根据里程记录进行判断，是否要走这条公路。
- 统计信息在**创建索引**或者执行**更新统计信息**命令时被创建，但是随着表中数据不断被插入和删除，或者表的统计信息由于不准确会变得失效，此时如果不更新统计信息，会导致检索突然间变得极其缓慢。
- 更新统计信息时，会占用系统资源，有时会产生加锁，所以要在系统负载较轻的时候使用



统计信息的更新方法

- **update statistics**命令可以用于更新统计信息
 - 参数**index_name**可以指定为某个索引的**key**列更新统计信息。
 - ✓ 例: `update index statistics users idx_user`
 - 参数**column_name**可以指定为某一特定列更新统计信息（这列通常没有被索引）
 - ✓ 例: `update statistics users (user_id)`
 - 参数**using step**可以指定更新统计信息时采样值的个数，缺省为**20**，而对于较大的表建议提高采样值（**Sybase**中文官方文档中叫做“直方图梯度”）
 - ✓ 例: `update index statistics users idx_user using 40 values`
 - 命令**update all statistics**用于为所有的列创建统计信息，但并不推荐
 - ✓ 例: `update all statistics users`



统计信息更新策略

- 要建立**定期更新**统计信息的机制
- 对于数据量较大的表，要**设定采样参数**，以获取更准确的统计信息。但也不是采样参数越大越好
- 对于那些数据量比较大的表，某一行没有添加到索引中，但是作为画面的查询条件经常被使用，应该**指定此列创建统计信息**



关于存储的优化

- 碎片
- 方法
- 秘技



碎片的产生

- 一个数据表，如果不断的进行数据的插入、删除和更新，会导致表数据和索引的存储产生碎片。（就像Windows系统的文件碎片一样）
- 碎片的产生会导致在一个页面上存放的数据是分散的，在检索时就会需要读取更多的页面，产生更多的IO，导致性能的下降
- 因此需要定期的维护，以减少碎片的产生



维护碎片的方法

- **reorg rebuild tablename [indexname]**命令用于重组表和索引的存储，如果添加了**indexname**参数则仅重组索引
 - 例： **reorg rebuild users**
- 在执行此命令前，强烈建议进行**bcp**备份，而实际上**Sybase**工程师的建议是...
- 执行此命令时，需要与表和索引相同的额外空间



统计信息的确认方法

■ **optdiag statistics**

- 可以确认表的统计信息
- 还可以确认表和索引的空间使用情况，当使用比率过低时，应考虑重建表或索引来重新填充数据

■ **例：** `optdiag statistics hq.lc.sales_promotion_detail -UXXXX -PXXXX -SXXXX -JXXXX`



关于temp数据库

- 概念
- 优化



temp数据库的基本概念

- 安装Sybase时缺省被分配到master库的数据文件中，安装后需要**立即手工创建更大的空间**
- temp库被所有其他的数据库共同使用，**用于创建临时表**
- temp库在下面情况下被使用
 - Order by (不同于索引顺序时)
 - Group by
 - Distinct
 - Create table #temp_table_name
 - 复杂的表关联查询
- temp库如果被**写满**，将导致用户提交的请求无法被相应的严重后果，一旦发生就需要重新启动数据库（带nowait参数）



temp库的优化

- temp库由于在处理请求时经常被系统暗自调用，所以尽可能将其数据文件与其他数据库的数据文件**分开磁盘存放**，以减少IO争用
- 如果数据缓存足够大，应该**将temp库单独绑定缓存**。因为temp的缓存使用与其他库完全相反，其他库被缓存调用时，会希望尽量的驻留在缓存中以提高命中率，而temp库恰恰相反，用过一次就希望被淘汰掉，如果放在一起，会导致整体的缓存命中率降低。但是如果整个的数据缓存只有**1-2G**则关系不大



索引的基本概念

- 查询时是否能使用到索引，对检索性能起到至关重要的作用
 - 查询时尽量能利用索引
- 索引并非越多越好，索引会：
 - 降低插入、修改、删除表时的速度
 - 占用空间，频繁的插入删除会产生索引碎块
 - 更新索引时，由于被加锁，可能对并发性能产生影响，甚至造成死锁
- 索引建立：
 - **索引的第一列最重要**，一般是查询中where语句必然指定的列
 - 索引列不宜指定太多，一般超过3列就没有太大意义了
 - 除了第一列，索引列的排序也会对检索性能产生影响



索引的种类

- 聚簇索引（**APL**）

- 数据是按照索引次序进行物理排序的，并且数据页是索引的叶子级，因此一个表最多只有一个聚簇索引
- 建立主键时缺省为聚簇索引
- 适用于的查询：
 - ✓ 范围查询
 - ✓ **Order**对应的列

- 非聚簇索引

- 非聚簇索引与行的物理顺序无关，一个表可以有多个非聚簇索引
- 只有非聚簇索引的表叫做“堆”
- 适用于的查询：
 - ✓ 点查询
 - ✓ 单列、单功能查询
 - ✓ 覆盖查询(索引覆盖)



索引覆盖

■ 概念

- 查询的结果集中的所有列都包含在索引中时，查询将不访问数据页，只访问索引列，因此产生较少的IO，提高检索效率

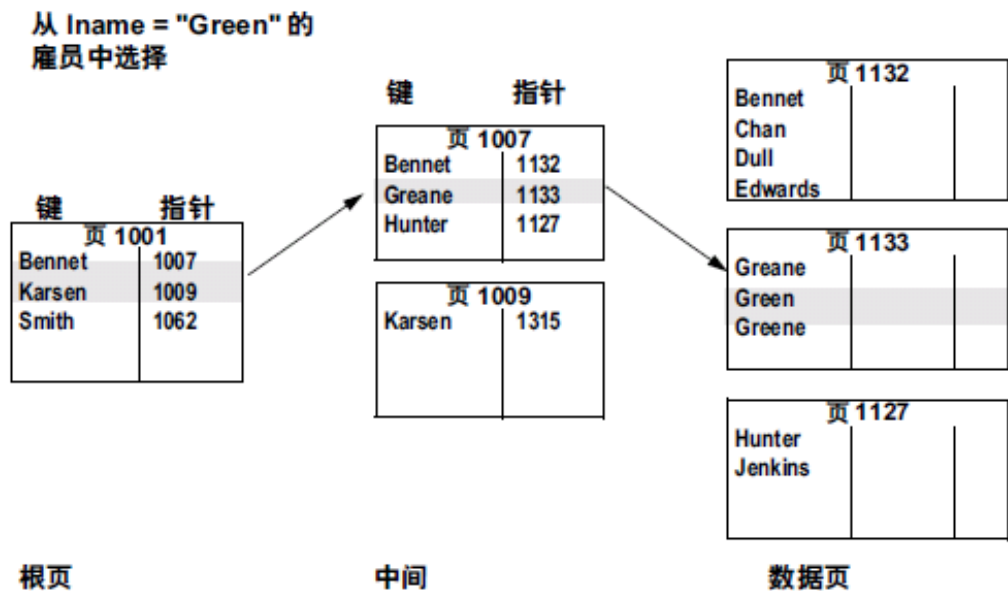
■ 注意

- 适用于行很宽，但是经常查询的列数量较少的表
- 不应将过多的列放入索引中，否则起不到减少IO的作用

索引存储-聚簇索引

■ 查询举例

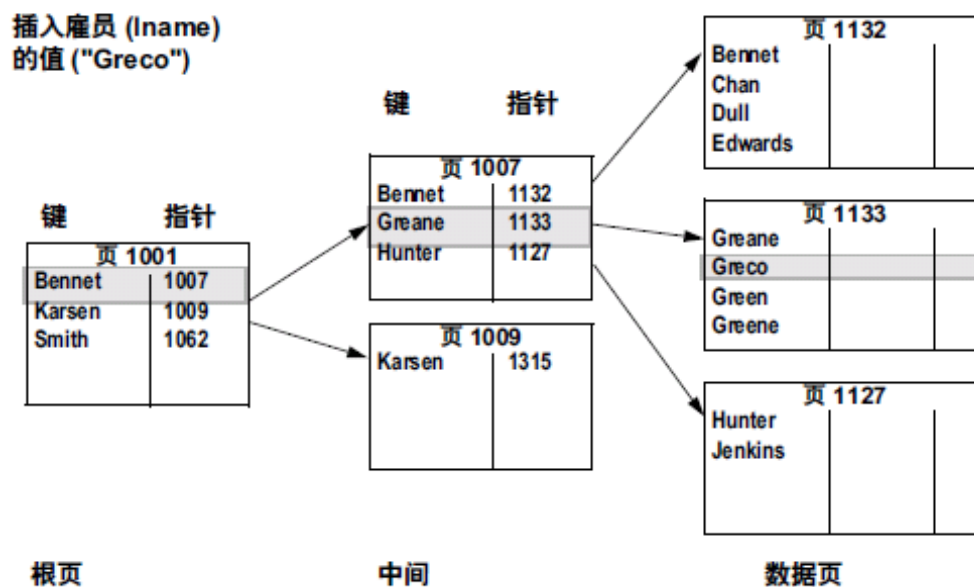
图 9-1: 使用集群索引在 *allpages* 锁表中选择一行



索引存储-聚簇索引

■ 插入举例

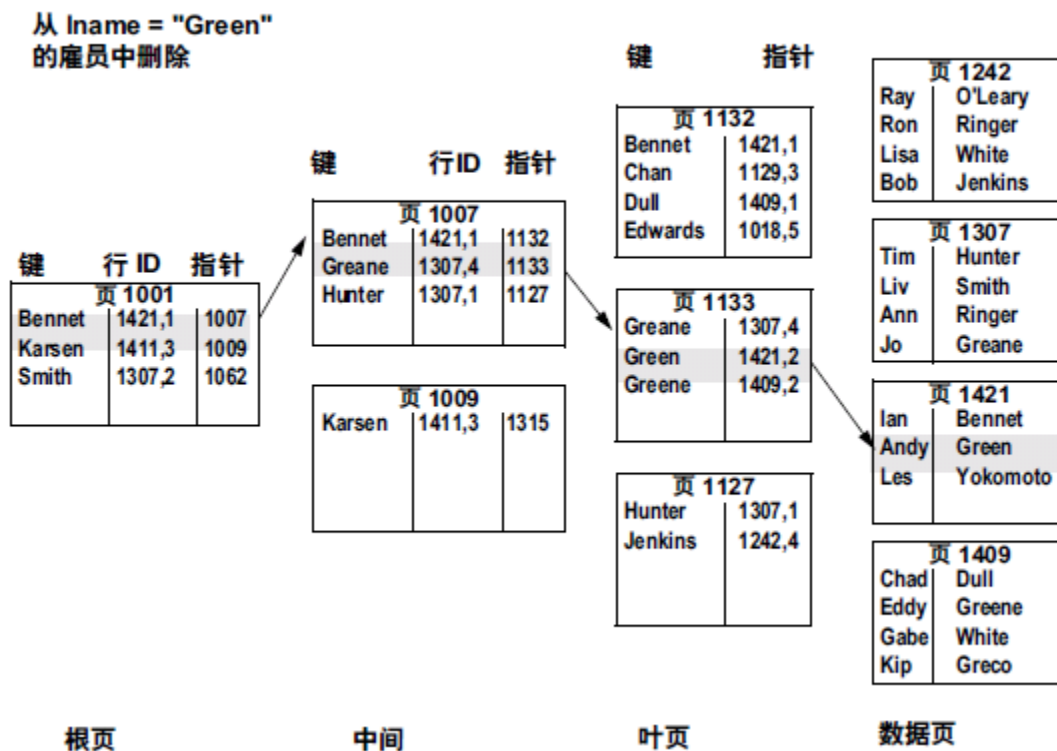
图 9-2: 向带有集群索引的 *allpages* 锁表中插入一行



索引存储-非聚簇索引

■ 查询举例

图 9-8：使用非集群索引选择行

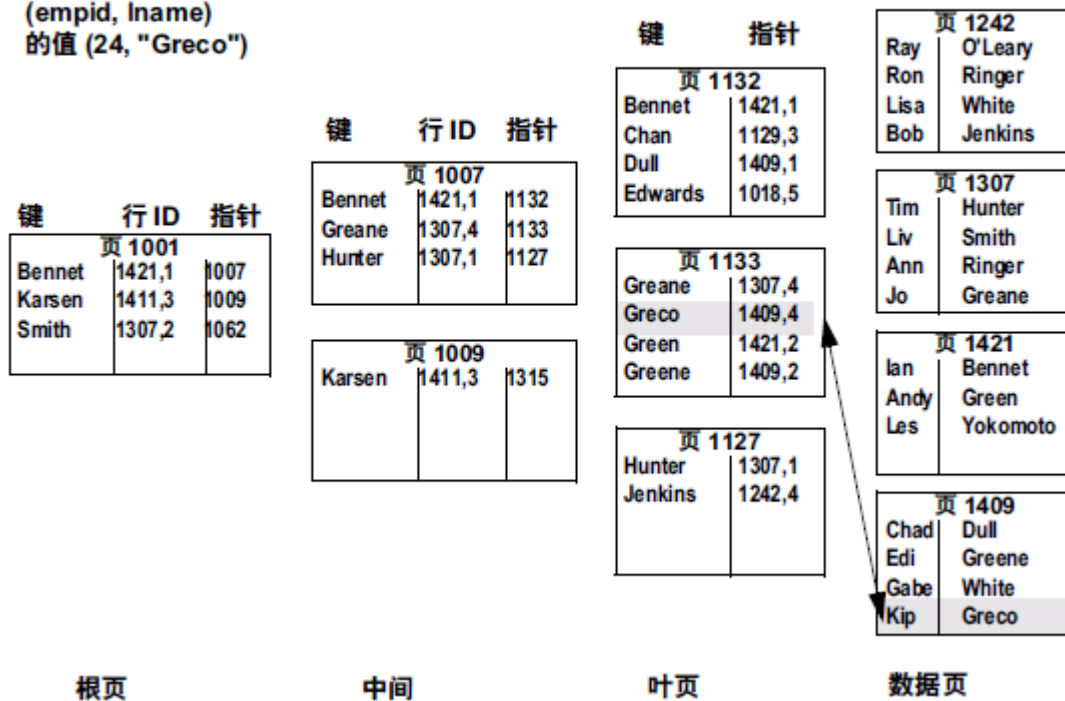


索引存储-非聚簇索引

■ 插入举例

图 9-9：对带有非集群索引堆表的插入操作

插入雇员
(empid, lname)
的值 (24, "Greco")





何时能用到索引

例：表TBL有A、B、C、D、E、F、G列

索引 idx_tbl 的列为： A、B、C

查询语句：select * from tbl where XXX=??? And XXX=??? ...

以下为各种查询条件的组合时，是否能用到索引的结果：

A='001'	B='2008/01/31'	C='20002313'	D=204	用到索引？
	有			NO
	有	有		NO
有				YES
有	有			YES
有		有		YES
有	有	有		YES
有	有	有	有	YES



何时能用到索引

应用例：

查询DTS表，假设DTS的索引是dts_store_id, dts_date

如果画面上没有指定按店铺查询，则画面生成的SQL文可能是：

```
select count(*) from lc.daily_transaction_summary  
where dts_date>='2007-12-01'  
and dts_date<='2007-12-31'
```

对DTS表进行了表扫描，用时**21**秒

如果将SQL文修改为：

```
select * from lc.daily_transaction_summary  
where dts_date>='2007-12-01'  
and dts_date<='2007-12-31'  
and (dts_store_id='001' or dts_store_id='002' or dts_store_id='003' or  
dts_store_id='004')
```

对DTS表运用了索引，用时**3**秒



何时能用到索引（片面结论）

■ 结论：

- **Where**子句中必须要包含索引第一列才能用到索引
- 是否还用到的索引的其他列来查询，还是会影响查询效率，但是远没有是否用到索引产生的影响大。
因此：**索引首列的设定最重要**
- **Where**子句中，包含了索引以外的列并不影响到是否使用索引（但是是否有高效的统计信息会对性能带来帮助，帮助的大小根据情况而定）
- **Where**语句中，各个条件的先后顺序没有关系



性能调查工具

■ `set showplan on/off`

- 用于最简单的查询执行计划
- 一般与`set noexec on/off`一起使用
- 应该关注的内容：
 - ✓ 是否产生了表扫描
 - ✓ 是否用到了正确的索引
 - ✓ 表的内外顺序是否合理（尽管有时看上去不合理）

■ `dbcc traceon/traceoff(3604,302,310)`

- 可以用来分析问什么优化器会执行`showplan`所显示的执行计划和开销统计等，但是很容很多，不容易看明白，非精通则不建议使用

showplan举例

查询文:

```
select
dts_date,
dts_article_id,
abi_article_name,
ac_category_id_user,
dts_current_cost,
dts_cost_adjust
from
lc.daily_transaction_summary,
lc.article_basic_info,
lc.article_category
where
dts_article_id=abi_article_id and
abi_category_id=ac_category_id and
isnull(dts_cost_adjust,0)<>0 and
dts_date>='2008/10/01' and
dts_date<='2008/10/31'
```

```
2> set showplan on
3> set noexec on
4> go
```

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1

The type of query is SELECT.

FROM TABLE

lc.daily_transaction_summary

Nested iteration.

Table Scan.

Forward scan.

Positioning at start of table.

Using I/O Size 4 Kbytes for data pages.

With LRU Buffer Replacement Strategy for data pages.

FROM TABLE

lc.article_basic_info

Nested iteration.

Index : article_basic_info_x

Forward scan.

Positioning by key.

Keys are:

abi_article_id ASC

Using I/O Size 4 Kbytes for index leaf pages.

With LRU Buffer Replacement Strategy for index leaf pages.

Using I/O Size 4 Kbytes for data pages.

With LRU Buffer Replacement Strategy for data pages.

FROM TABLE

lc.article_category

Nested iteration.

Index : article_category_x

Forward scan.

Positioning by key.

Keys are:

ac_category_id ASC

Using I/O Size 4 Kbytes for index leaf pages.

With LRU Buffer Replacement Strategy for index leaf pages.

Using I/O Size 4 Kbytes for data pages.

With LRU Buffer Replacement Strategy for data pages.

←发生表扫描
最外层表

←用到索引
中间层表

←用到索引
最内层表



对ASE查询优化器的简单理解

- ASE通过优化器确认访问数据库的最优方法
 - 是否利用索引，还是表扫描
 - 连接表的顺序（内表和外表），以及连接的类型
 - 使用多大的IO输出
 - 如何使用并行查询
- 优化器对查询产生的执行计划不是一成不变的
 - 当查询条件变化时
 - 当统计信息变化时（表的数据件数发生变化时）



关于索引的建议-1

- 对于“**基础主档**”类的表
 - 特点：
 - ✓ 表的数据一经建立，很少发生变化
 - ✓ 表的行数较少，一般在数千件以内
 - ✓ 经常被很多查询所引用
 - ✓ 一般有一个字段是**ID**字段，是唯一主键
 - ✓ 例如：商品单位主档表、商品分类主档表
 - **对于这个唯一主键应该建立非聚簇索引**（因为主要是作为别的表的外键进行点查询）
 - 而且一般来说，不再需要建立其他的索引
 - 可能的话，添加附加列，以实现“索引覆盖”。
 - ✓ 例如：每个画面的查询几乎都用到了**BCR**和**BUR**两个表，但是几乎没有影响到性能，就是因为实现了索引覆盖
 - 虽然有主键索引，但是在查询计划中，很可能是在进行表扫描。原因是表的数据量很少，即使进行表扫描也没有太大开销



关于索引的建议-2

- 对于“**业务主档**”类的表
 - 特点：
 - ✓ 是业务类的关键主档，随着业务的进行，主档不断的被维护
 - ✓ 表的行一般比较多，但不是非常多，通常在数千到数万件
 - ✓ 表经常被很多查询所引用
 - ✓ 一般有一个字段是**ID**字段，是唯一主键
 - ✓ 表的主键列以外，有些是外键**ID**，需要参照“基础主档”类的表来获取**ID**代表的基础信息
 - ✓ 表的主键列以外，还有一些状态列，常用于限制主档的选择范围
 - ✓ 例如：商品主档、供应商主档、专柜主档
 - 对于这个**唯一主键应该建立非聚簇索引**（因为主要是作为别的表的外键进行点查询）
 - 对于外键**ID**的列或者是状态列，如果存在经常需要查询的**where**语句中出现，可以考虑添加这一列的**非聚簇索引或者用于覆盖的索引**（当该外键**ID**的重复值过多时索引效率可能并不高）
 - ✓ 例如：对ABI的abi_category_id添加索引后，在画面上按照分类查询商品主档一览就大幅度提高了性能



关于索引的建议-3

- 对于“**业务数据**”类的表

- 特点:

- ✓ 是随着业务的进行，几乎每天都产生新数据的表
 - ✓ 表的行一般随日期增长而不断变化，整体件数很多，通常在数万件到数百万
 - ✓ 表数据经常在进行业务类的查询时使用
 - ✓ 通常是复合主键，其中大都是店铺、日期、商品、单据号等的组合
 - ✓ 有时作为业务单据的表，有系统产生的唯一单据号，但是很少在查询中使用
 - ✓ 在查询时，通常要关联“业务主档”来检索
 - ✓ 例如：促销变价档、进退货档、会员消费档
 - 这类表的主键，很可能是为了维护表数据的整合性而存在，在查询时并不一定会被用到
 - 通常需要对日期进行查询，建立首列是日期（或者是仅针对日期）的索引通常是必要的
 - ✓ 例如：正常变价查询画面在建立的变价生效日的索引后，查询从原来的几分钟缩短到了几秒钟
 - 除此之外，根据业务查询画面的需要创建合适的索引，但是应该尽可能的少。反之，设计画面时要考虑是否能用到现有的索引



关于索引的建议-4

- 对于“**信息分析**”类的表
 - 特点：
 - ✓ 是随着业务的进行，几乎每天都产生新数据的表
 - ✓ 表的行一般随日期增长而不断变化，整体件数很多，通常在数十万件到数百万
 - ✓ 表数据经常在进行信息分析查询时使用
 - ✓ 通常是复合主键，其中大都是店铺、日期、分类等的组合
 - ✓ 在查询时，通常要按日期查询才有意义
 - ✓ 例如：每日单品进销存档、每日分类进销存档
 - 这类表的主键，很可能是为了维护表数据的整合性而存在，在查询时常被用到，由于查询时，日期是必选项，所以日期作为主键的首列比较好
 - 这类表的数据的做成通常是在批处理，做成后一般不发生变化（删除除外）
 - 如果经常是范围查询，比如查询日期段内的销售，就应该考虑创建聚簇索引。但应该注意的是，插入表数据时，应先按照主键顺序排序，以减少页扩展，防止在插入操作时性能明显下降



画面设计与索引设计

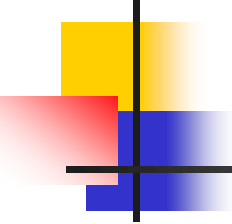
- 对现有表设计新画面时
 - 要调查现有表的索引现状，确认新画面是否有可能用到现有索引
 - 由于查询条件的不同会影响执行计划，设计画面时应该考虑：
 - ✓ 哪些是必须查询的条件（例如：必须按照店铺查询），必须查询的条件应该是索引的首列
 - ✓ 查询条件是否应该进行限制（例如：最多查询1个月数据），以保证不会随着用户输入条件的变化使查询计划发生本质改变
 - ✓ 根据现有的表的数据规模，估计查询返回的数据行数
 - 如果现有索引无法满足查询条件，不用索引会导致性能很差，就应该考虑创建新的索引，但是上面的考虑项仍旧要考虑，并且非聚簇索引的个数不应该过多（一般不超过5个）
 - 根据已有的估计，要按照各种可以输入的查询条件进行组合（原则上只需要递加式的组合），进行简单的测试，看执行计划是否合理，查询性能是否理想
 - 对于无论如何都会很慢的画面：
 - ✓ 考虑采用批处理做成数据源的方式解决，但是要考虑数据源的开发难度、复用性、维护难度、重建可能性等因素
 - ✓ 与客户充分沟通，增加画面限制或者减少画面显示项目，使现有索引能充分利用，保证性能在可控范围内



画面设计与索引设计

■ 开发新应用创建新数据源时

- 表的主键是什么，是否仅仅为了实现数据唯一性而创建，还是可以用于查询
- 表属于哪一种类型，应该遵循什么规则创建索引
- 画面会从什么角度查询数据，哪些条件预计是必须输入的，这个条件应该是索引的首列
- 此表预计的数据量是多少
- 此表的行宽是多少，是否有可能做成索引覆盖
- 此表今后有可能会有的应用是什么
- 此表配合的画面的查询性能要事先估计，如果之后开发出来的结果与现有状况不符要进行调查
- 如果是设计频繁访问的关键画面，要进行手工增幅，进行性能测试，以保证长期的使用安全性



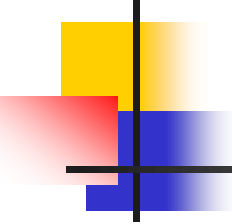
SQL文书写注意点

- **where** 条件中不要针对列使用函数，否则有可能无法使用索引

➤ 例如：ASAH表有针对**asah_date**创建索引，但是下面两个查询，表面上没有太大区别，实际上第二个检索会产生全表扫描

```
SQL1:  
select *  
from lc.article_stock_adjust_header  
where asah_date='2008/10/01'
```

```
SQL2:  
select *  
from lc.article_stock_adjust_header  
where convert(char, asah_date, 112)='20081001'
```

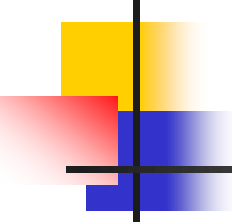


SQL文书写注意点

- 复合索引的非前导列做条件时，基本没有起到索引的作用
 - 例如：DTS表的复合主键索引是：dts_store_id, dts_date, dts_article_id, 下面的语句不会用到索引

```
SQL1:  
select *  
from lc.daily_transaction_summary  
where dts_date='2008/10/01'
```

- SQL 文不宜写得太复杂，当查询要关联很多表时，可以先从大表生成临时表，再用临时表与其他表相关联生成最终结果集



SQL文书写注意点

- 使用**like**语句时，应该尽量避免将**%**写在最左边，这样会导致无法使用索引
 - 例如：**name**字段建有索引，但是下面SQL文

```
SQL1:  
select *  
from member  
where name like ' %建华 '
```